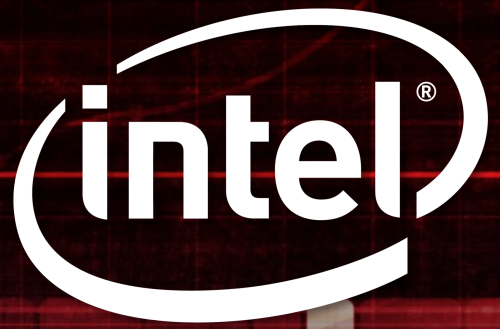


Intel Data Parallel C++

Jeff Hammond
Intel

Outline

- Hardware diversity and open standards
- Intel oneAPI and Data Parallel C++
- Khronos SYCL
- oneAPI libraries and usage info
- Kokkos and RAJA update
- Multi-GPU programming SYCL

Problem Statement

Diversity and complexity in computer architecture has been growing continuously since the year 2000 and there is no indication that programming is going to get any easier any time soon.

Even with architectural families, there are differences in how vendors implement processors, both with software and hardware.

While performance tuning is architecture-specific and often microarchitecture-specific, programmers are most productive when tuning working code, as opposed to porting code then tuning it.

US-DOE Exascale Systems (2021+)

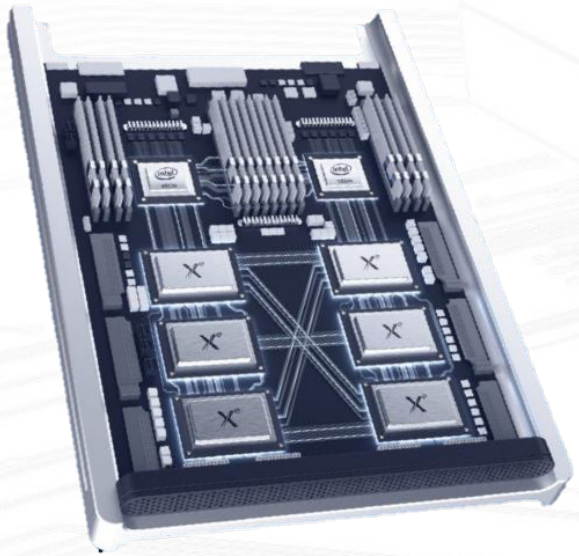
Neither of these systems is has a many-core CPU or an NVIDIA GPU…

What programming model(s) take a developer from 2012 to 2022?

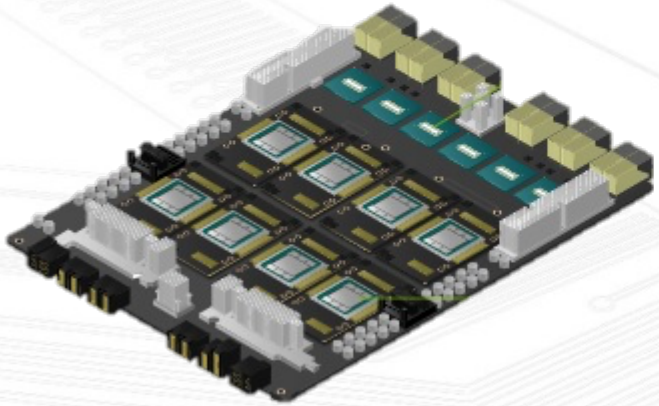
intel

GPU Supercomputers are Multi-GPU Systems

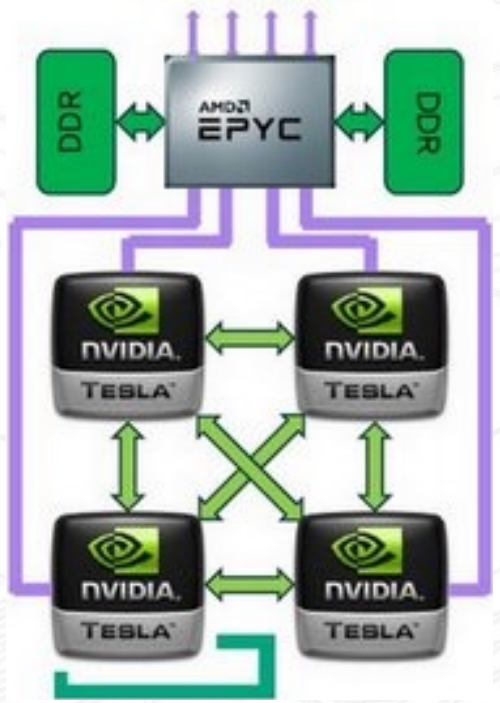
Argonne Aurora: 2 CPU & 6 GPU

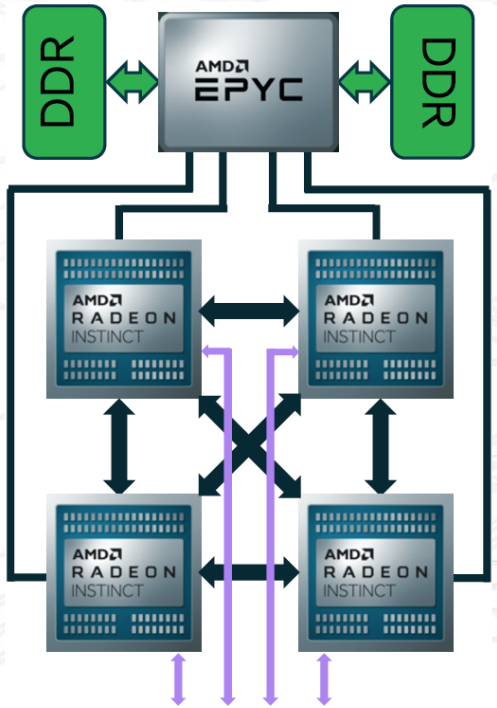


NVIDIA HGX A100: 2 CPU & 8 GPU



NERSC Perlmutter and ORNL Frontier: 1 CPU & 4 GPU





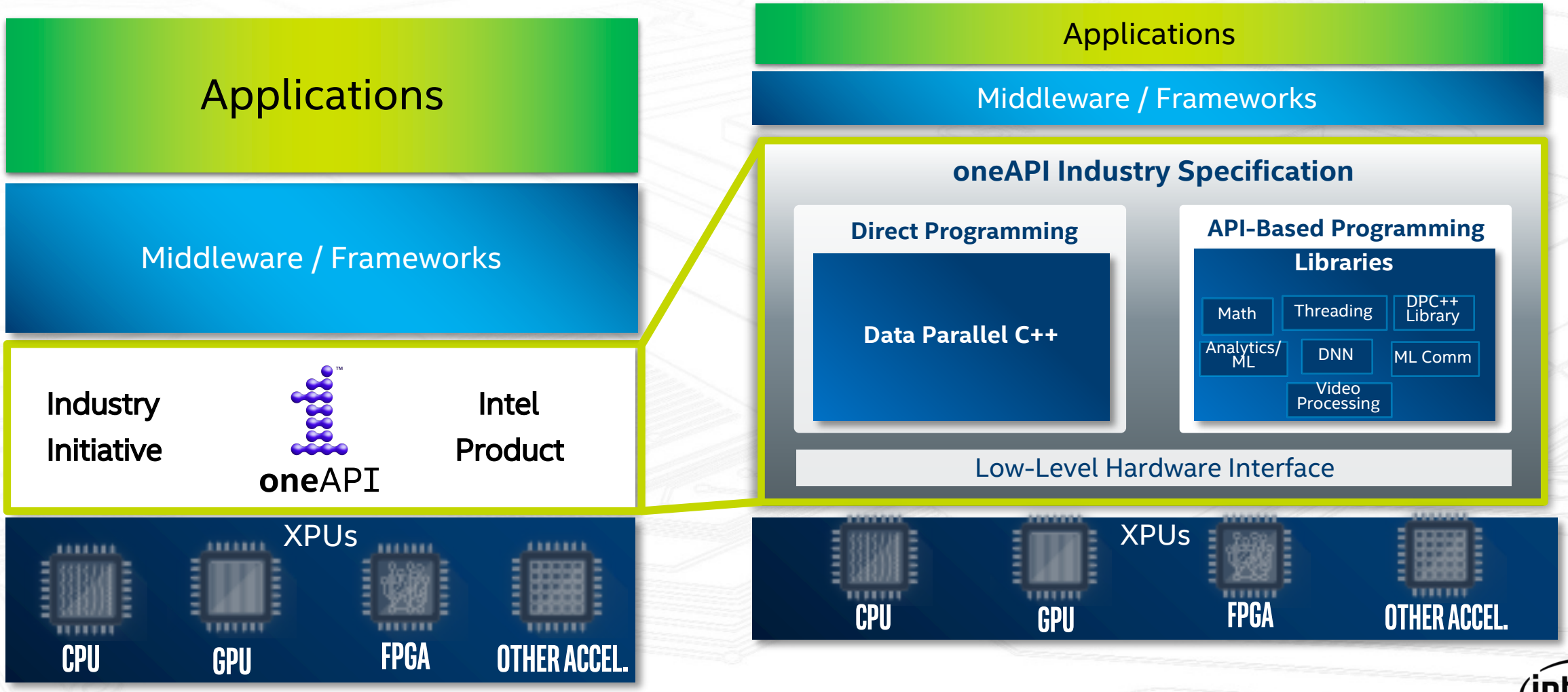
https://www.servethehome.com/wp-content/uploads/2019/11/SC19-Intel-DoE-Aurora.jpg

https://www.enterpriseai.news/2020/05/20/amd-epyc-rome-tabbed-for-nvidias-new-dgx-but-hgx-has-intel-option/

https://www.hpcwire.com/2020/03/11/steve-scott-hpe-cray-blended-product-roadmap/

intel

INTEL ONEAPI



Visit oneapi.com for more details

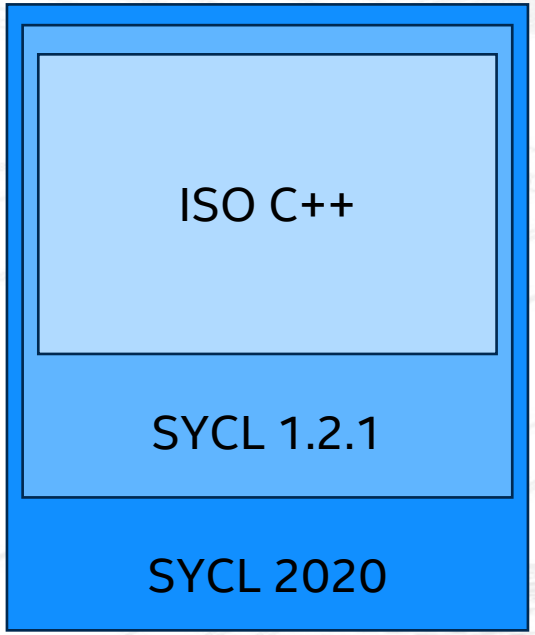
Khronos SYCL 2020 and Data Parallel C++

Intel DPC++ is a Clang-based open-source compiler for ISO C++ and Khronos SYCL.

The SYCL 2020 provisional specification includes a number of important improvements to SYCL 1.2.1:

- Unified Shared Memory (USM)
- Reductions
- Subgroups
- In-order queues

Intel continues to work with the SYCL community to bring additional language features into the standard.



Intel's extensions – both the documentation and the implementation source code - are currently available on GitHub: https://github.com/intel/llvm/

Why SYCL?

OpenCL has a well-defined, portable execution model, but is considered too verbose by application programmers and lacks good C++ support.

SYCL is based on purely modern C++, which allows it to support heterogeneous accelerators within a single-source model.

SYCL parallelism is similar to TBB and the C++ STL while giving users explicit control over hardware resources when they want it.

WHY SYCL?

OpenCL has a well-defined, portable execution model, but is considered too verbose by application programmers and lacks good C++ support.

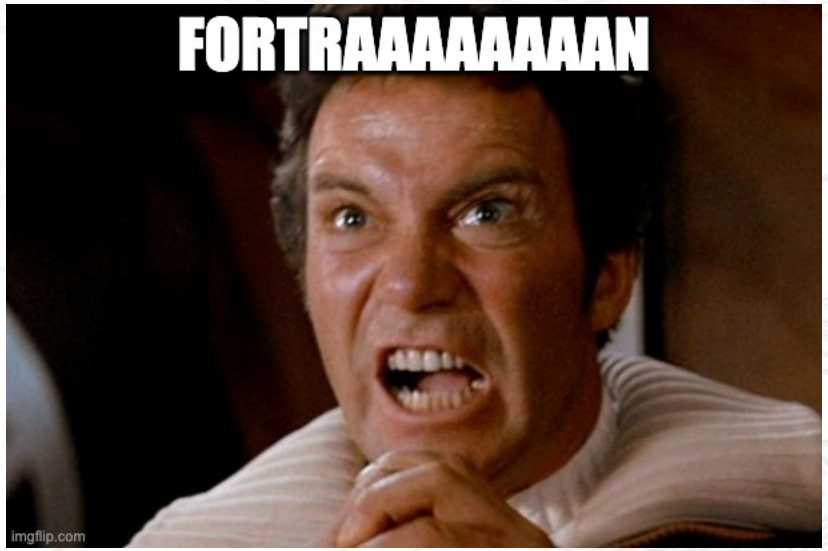
SYCL is based on purely modern C++, which allows it to support heterogeneous accelerators within a single-source model.

SYCL parallelism is similar to TBB and the C++ STL while giving users explicit control over hardware resources when they want it.

SYCL is the first standard programming model designed for heterogeneous programming with modern C++

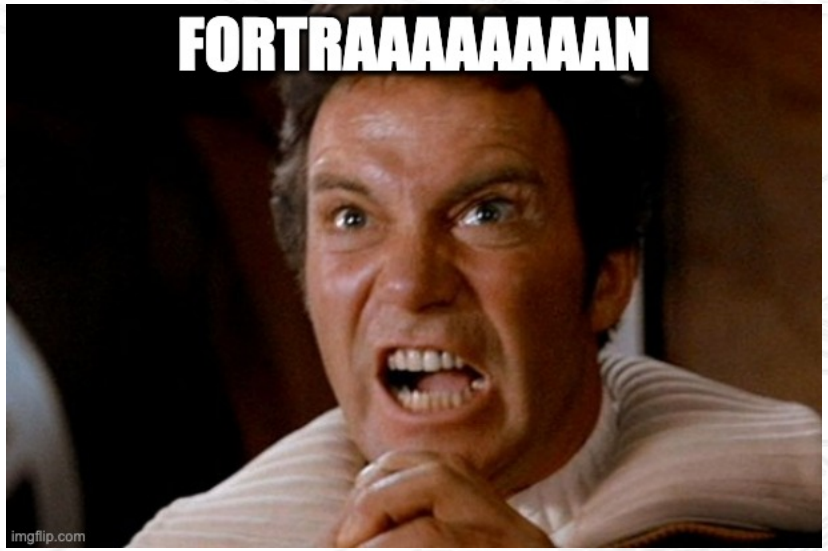
Why Not SYCL?

- If you do not like modern C++, you will not like SYCL.
- If you want access to low-level hardware-specific features that are only accessible using a vendor-provided model, SYCL may not meet your needs.
- Unlike OpenMP, some hardware vendors are not supporting OpenCL/SPIR-V.



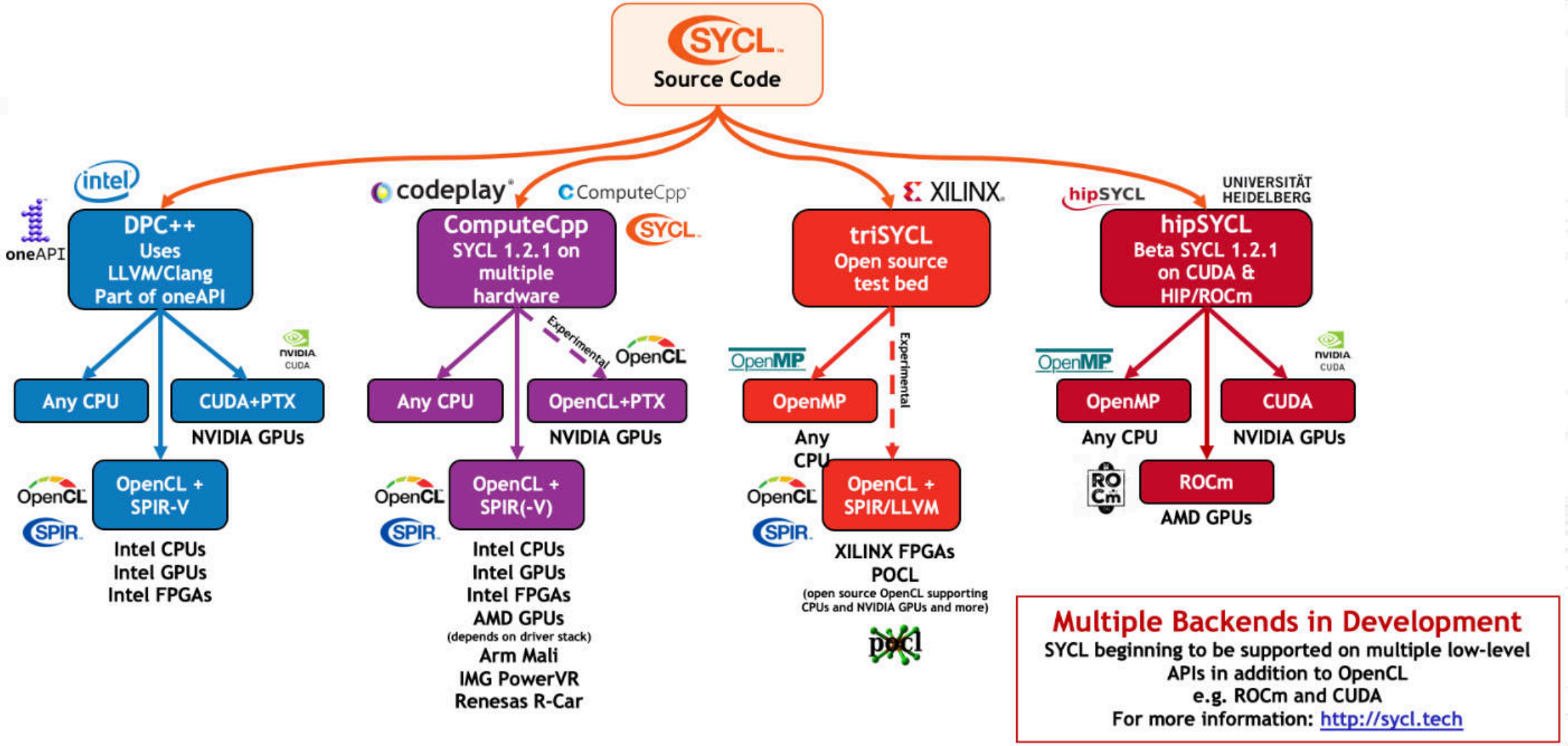
Why Not SYCL?

- If you do not like modern C++, you will not like SYCL.
- If you want access to low-level hardware-specific features that are only accessible using a vendor-provided model, SYCL may not meet your needs.
- Unlike OpenMP, some hardware vendors are not supporting OpenCL/SPIR-V.



The Intel oneAPI HPCKit includes OpenMP target for GPUs, including support for MKL.

SYCL Ecosystem as of June 2020



https://www.khronos.org/assets/uploads/apis/2020-05-sycl-landing-page-02_2.jpg

SYCL Platform Portability Measurements

- Authors:
  Tom Deakin and Simon McIntosh-Smith
  of the University of Bristol
- Paper:
  https://dl.acm.org/doi/abs/10.1145/3388333.3388643
- Video:
  https://www.youtube.com/watch?v=5W6SsreZ3ew
- Code: https://github.com/UoB-HPC/BabelStream

The low SYCL/OpenCL performance on Intel Xeon processors is a known implementation issue in Intel OpenCL. It is not a fundamental limitation and will be fixed in the future.

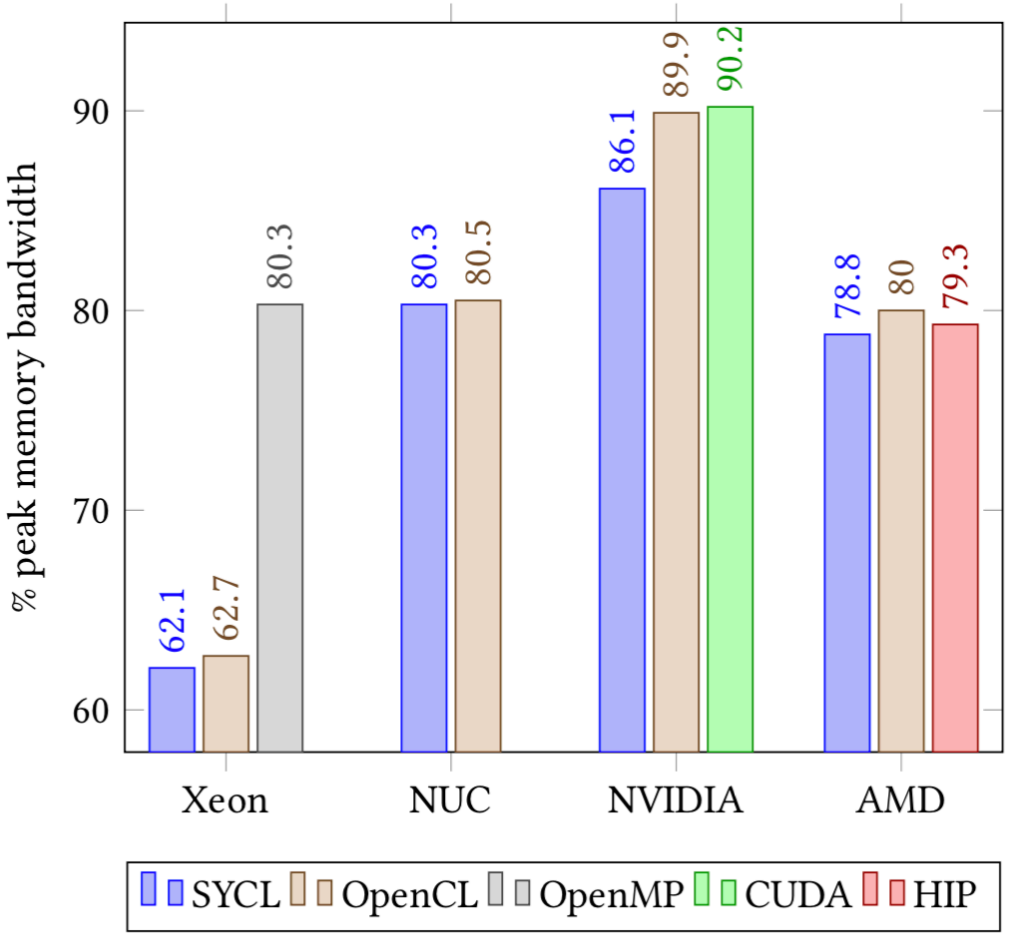


Figure 1: BabelStream Triad results

SYCL Platform Portability Measurements

- Authors:
  Brian Homerding and John Tramm of Argonne National Laboratory
- Paper:
  https://dl.acm.org/doi/abs/10.1145/3388333.3388660
- Video:
  https://www.youtube.com/watch?v=-xzuFLZ64W0
- Code:
  https://github.com/homerdin/RAJAPerf/tree/sycl

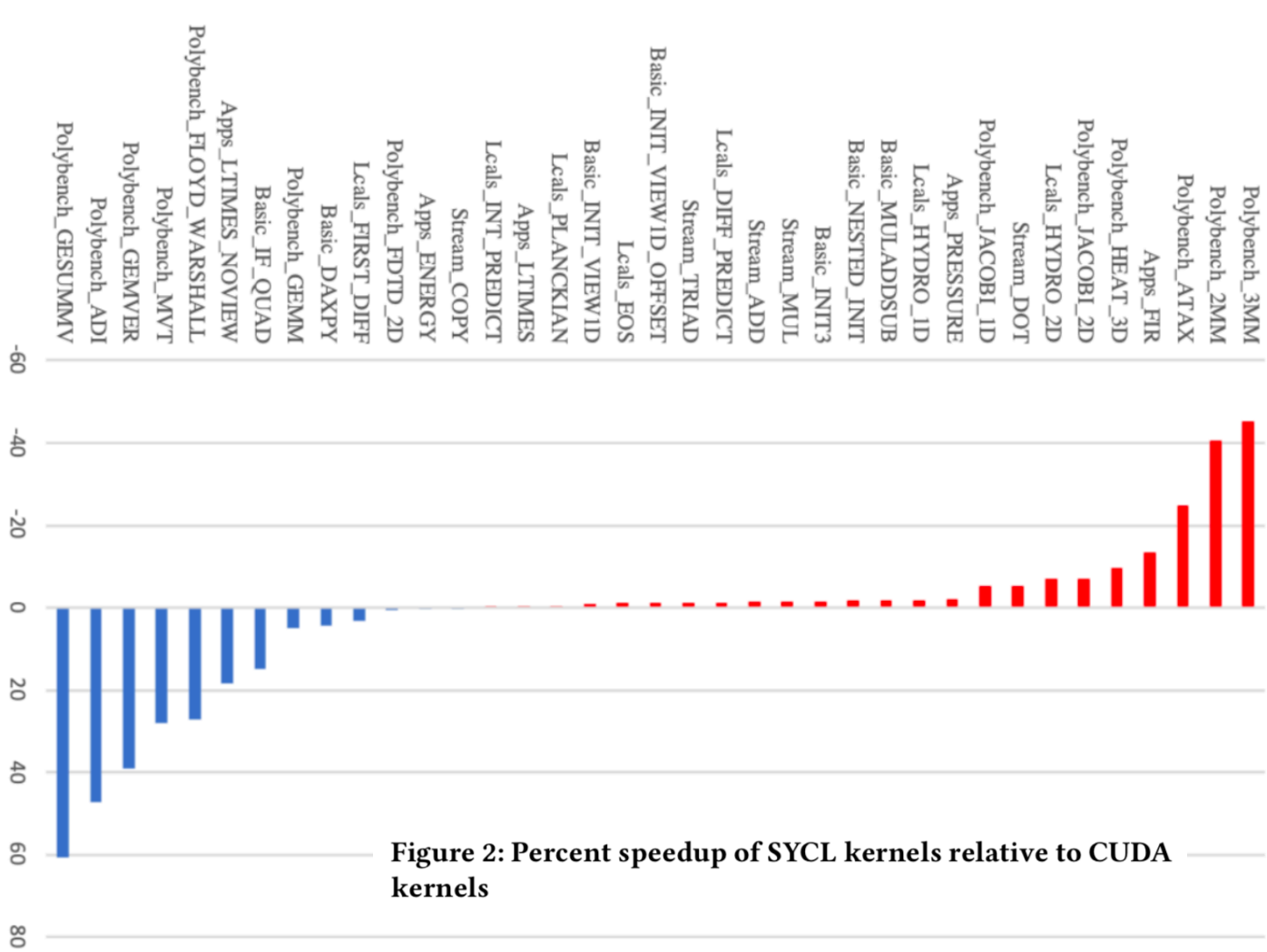


Figure 2: Percent speedup of SYCL kernels relative to CUDA kernels

PROGRAMMING IS ALSO ABOUT THE CODE YOU DON'T WRITE!

The Story of a Man Who Outsourced His Work to China so He Could Watch Cat Videos All Day

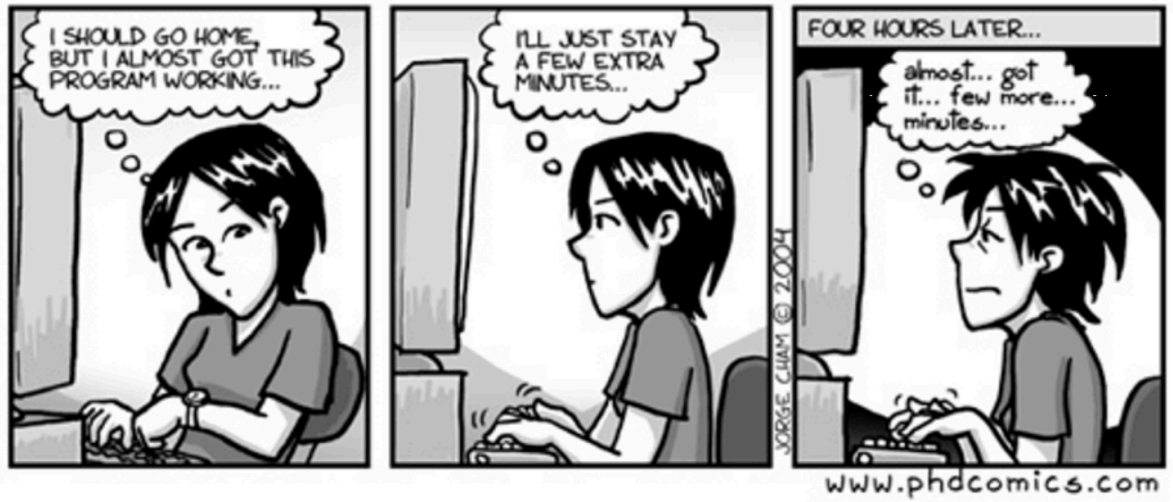
BUSINESS INSIDER By Megan Rose Dickey | Business Insider – Wed, Jan 16, 2013 8:57 AM EST

Email Recommend 25.5k Tweet Share +1 Print

The Story of a Postdoc Who Outsourced Her Programming to Libraries so She Could Do Science All Day

BUSINESS INSIDER By Megan Rose Dickey | Business Insider – Wed, Jan 16, 2013 8:57 AM EST

Email Recommend 25.5k Tweet Share +1 Print



intel

oneAPI Libraries

- oneDPL: C++ standard library functions, including GPU parallel STL
- oneMKL: math library for Intel CPU and Intel GPU
  - CodePlay contributed CUBLAS support
- oneDNN: Deep Neural Network Library (was MKL-DNN)
  - Supports a variety of non-Intel processors already
- oneCCL: Collective Communication Library (was MLSL)
- oneDAL: Data Analytics Library (was DAAL)
- oneVPL: Video Processing Library

https://spec.oneapi.com/versions/latest/index.html

Learn more about oneAPI

- oneAPI specifications https://www.oneapi.com
- Intel oneAPI implementation https://software.intel.com/en-us/oneapi
  - Apt, Yum, Zypper installation on Linux
  - Docker
  - Traditional online and offline binary installers for Linux and Windows
  - DevCloud: https://intelsoftwaresites.secure.force.com/devcloud/oneapi
    - DevCloud includes CPU, GPU and FPGA hardware…
- Tutorials and sample code
  - https://github.com/jeffhammond/dpcpp-tutorial
  - https://github.com/alcf-perfengr/sycltrain
  - https://github.com/oneapi-src/oneAPI-samples
  - https://software.intel.com/content/www/us/en/develop/articles/brightskies-experience-using-oneapi-for-reverse-time-migration.html

K&R C++ Enabling in oneAPI

- Active development of SYCL/DPC++ and OpenMP target backends driven by DOE ECP and Aurora Early Science Projects.
- SYCL 2020 spec added key features required by Kokkos/RAJA
  - Unified shared memory (pointer-based) and allocators
  - Unnamed lambda support
  - Reductions
- Application development in progress using
  - RAJA/Umpire SYCL/DPC++ backend
  - Kokkos OpenMP target backend

Detect All the GPUs

```
private:
  std::vector<sycl::queue> list;
public:
  queues(void) {
    auto platforms = sycl::platform::get_platforms();
    for (auto & p : platforms) {
      auto devices = p.get_devices();
      for (auto & d : devices ) {
        if ( d.is_gpu() ) {
          list.push_back(sycl::queue(d));
        }
      }
    }
  }
```

Assumptions and logic to ensure devices are in the same context are hidden…

https://github.com/jeffhammond/PRK/blob/dpcpp-multi-gpu-transpose/Cxx11/prk_sycl.h

SYCL 2020 Data Models

- Buffers are wonderfully opaque but makes reasoning about sharing hard.
- USM shared data moves between host and device (d2d is not portable).
- USM device data does not migrate – start here.

Allocation Type	Initial Location	Accessible By		Migratable To	
device	device	host	No	host	No
		device	Yes	device	-
		Another device	Optional (P2P)	Another device	No
host	host	host	Yes	host	-
		Any device	Yes (~PCIe)	device	No
shared	host / device / Unspecified	host	Yes	host	Yes
		device	Yes	device	Yes
		Another device	Optional (P2P)	Another device	Optional

Device Allocation

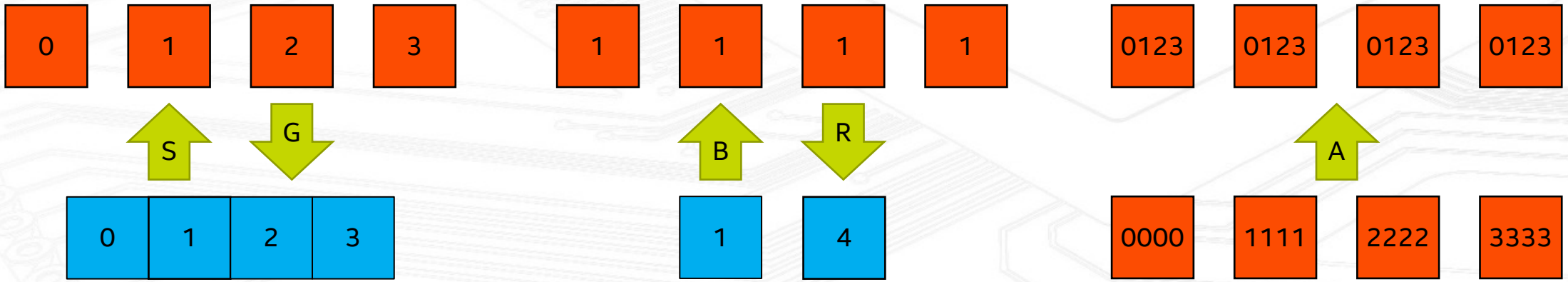
```
template <typename T>
void allocate(std::vector<T*> & device_pointers, size_t num_elements)
{
    for (const auto & l : list | boost::adaptors::indexed(0) ) {
        auto i = l.index();
        auto v = l.value();
        device_pointers[i] = sycl::malloc_device<T>(num_elements, v);
    }
}

// Placement new is your friend if you want to allocate nontrivial
// types in GPU, e.g. Quicksilver.
// std::pmr also works (there should be an example in the DPC++ repo).
```

https://github.com/jeffhammond/PRK/blob/dpcpp-multi-gpu-transpose/Cxx11/prk_sycl.h

Design Strategy

Bulk-synchronous, host-controlled data movement:

- Broadcast to devices, Reduce from devices (replicated data)
- Scatter to device, Gather from devices (unique data)
- Device all-to-all (unique data)
- Add more later…



Implementation Details

- All of these operations can be implemented in naïve but portable O(N) ways.
- All of these operations can optimized with device P2P, etc.

```
template <typename T, typename B>
void scatter(std::vector<T*> & device_pointers,
             const B & host_pointer,
             size_t num_elements)
{
    auto bytes = num_elements * sizeof(T);
    for (const auto & l : list | boost::adaptors::indexed(0) ) {
        auto i = l.index();
        auto v = l.value();
        auto target = device_pointers[i];
        auto source = &host_pointer[i * num_elements];
        v.memcpy(target, source, bytes);
    }
}
```

https://github.com/jeffhammond/PRK/blob/dpcpp-multi-gpu-transpose/Cxx11/prk_sycl.h

Example Program

Trivial data-parallel compute

```
for (int g=0; g<np; ++g) {
    auto q = qs.queue(g);
    auto p_A = d_A[g];
    q.parallel_for( sycl::range<1>{N},[=] (sycl::id<1> i) {
        p_A[i] += 1;
    });
}
```

State

```
auto qs = queues();
auto np = qs.size();
auto h_A = prk::vector<T>(np*N, 0);
auto d_A = std::vector<T*>(np, nullptr);
```

Helper functions

```
qs.allocate<T>(d_A, N);
qs.free(d_A);
qs.scatter<T>(d_A, h_A, N);
qs.gather<T>(h_A, d_A, N);
qs.waitall();
```

https://github.com/jeffhammond/PRK/blob/dpcpp-multi-gpu-transpose/Cxx11/nstream-multigpu-dpcpp.cc

Multi-GPU SYCL

- Like MPI, SYCL is explicit about providing a handle to state.
  - MPI communicators, groups, requests, windows, etc.
  - SYCL platforms, devices, queues, contexts, etc.
- Explicit device handles make the task of multi-GPU programming easier.
  - Contrast: CUDA runtime API hides the device id and CUBLAS contexts don't capture it…
- If GPU compute is tightly coupled, build a data-centric abstraction to manage allocation, data movement, synchronization, and collective compute.
  - PETSc, Elemental and Global Arrays are good examples of this in the MPI plane.
- Load-store is not a good abstraction for most interconnects…

```
for (int i=0; i<ngpus; ++i) {
  check( cudaSetDevice(i) );
  check( cublasCreate(&contexts[i]) );
}
```

```
for (int i=0; i<ngpus; ++i) {
  check( cudaSetDevice(i) );
  check( cudaDeviceSynchronize() );
}
```

https://github.com/jeffhammond/PRK/blob/dpcpp-multi-gpu-transpose/Cxx11/dgemm-multigpu-cublas.cu

Multi-GPU SYCL

- The hard problems in multi-GPU programming are identical to those of multi-anything programming:
  - Data decomposition, load-balancing, debugging, etc.
  - Whether you use MPI or not, thinking like an MPI programmer will help.
- For balanced computation and symmetric performance, bulk-synchronous compute is fine, and makes parallel programming easy.
  - Bulk synchronous gets a bad name because of imbalanced computation and asymmetric performance (jitter), which is associated with uncoupled systems (nodes) and contended networks.
  - The data movement efficiency of optimized collectives can be worth quite a lot.
- For imbalanced computation, pre-compute as much of the work assignment as possible (inspector-executor model).


intel®